埼玉女子短期大学研究紀要　第6号　1995.03

数式処理と画像描画

Symbolic Computation and Plotting Functions

近藤祐史＊　三好善彦†　齋藤友克‡　竹島　卓§
Yuji Kondo　Yoshihiko Miyoshi　Tomokatsu Saito　Taku Takeshima

Visualization helps us to understand phenomenons, to see into the background, to discover a new question and so on. Usually, we use numerical method to draw graphs. We may research new approach using symbolical computation to draw graphs. In this paper, we propose a basic starting point to draw function curves.

1　はじめに

計算機を用いて様々な問題を解決しようとする場合、計算の結果を可視化することは、現象の理解、背景の洞察もしくは新しい課題の発見等のために欠くべからざるプロセスになっている。特に数式で与えられた問題の幾何的イメージを把握出来れば、解の存在範囲や交点の個数配置などが容易に見る事が出来、最終的な答えを導き出すことに役にたつ場合がある。

関数描画としては、基本的な出発点として大別して追跡法と全画面検索法がある。我々の基本となる方針は全画面検索法を前提として考察する。本稿では関数の描画に対する基本的な出発点を提起したい。

追跡法とは、関数値をいくつか計算し、順番に辿っていき、その軌跡として関数描画を行うものである。この場合の利点として全体の計算量が全面検索法に比べて大変少ないことがあげられるが、問題点としては関数の正確な描画をすることができないことがあるということである。対して全画面検索法は画面上の全ての点に対して関数値を計算し、その時の値を判定し関数描画を行う方法である。このため、計算量は追跡法と比較すると格段に増える。しかし、現在の計算機の性能を考えれば計算量に関するコストはそれ程問題に

＊詫間電波高専　kondoh@dc.takuma-ct.ac.jp
†埼玉女子短大　CXK02432@niftyserve.or.jp
‡上智大学理工・数学　saito@mm.sophia.ac.jp
§富士通　情報研　tak@iias.flab.fujitsu.co.jp

ならなくなっている。このことを考えれば、追跡法により正確な関数描画ができないことよりも、全面検索法により正確な関数描画が望まれるようになってくる。以上が、我々の方針が全面検索法を前提とする所以である。

2　数式処理と関数描画

計算機を用いて計算をする場合、その方法としていくつかの手段があるが代表的なものとして、直接数値を扱って計算をする数値計算と、数式を直接人間が扱うように計算する数式処理がある。

数値計算は計算速度が速いが、計算誤差が発生するという欠点がある。数式処理は計算誤差は発生しないが、計算速度が遅いという欠点がある。しかし、ここ数年の計算機の処理速度の向上を考えれば、計算速度に関する欠点は解決しつつある。本稿では、正確な関数描画を行うことを前提としているため、計算誤差が生じない数式処理を利用することとする。

2．1　数式処理システムとは

数値計算を行うことによって、計算結果を簡単に求めることができるが、この結果はある程度正確であっても必ず計算誤差が生じている。以前はこのような結果であっても必要な精度が得られていたが、より高精度な結果を得ようとすると、アルゴリズム的に数値計算の限界につきあたる場合がある。

なぜなら、計算機で数値を表現をしようとするとき、数値は常に近似値として表現されることにより生ずる誤差が発生するからである。（計算機で表現できる大きさ内で表現するために、はみ出した部分は切り捨てを行う。この時、切り捨てられた部分が誤差となる。）

さらに、計算を繰り返すうちにアルゴリズムに内在する誤差が生ずる。（通常、数値計算のアルゴリズムは解析的なアルゴリズムにより構築されている。そのため、手続きは、無限回のステップにより真の解を求めるが、計算機のアルゴリズムとしては十分な精度が求まったときに手続きを打ち切る有限回のステップにより求める。）

一方数式処理システムは、数値を厳密に表現し、代数的なアルゴリズムを利用することにより誤差は発生しない。しかし、この方法が最善かと言うと、数値計算よりも格段に計算量が増大し、解決できる問題の範囲も限られたものとなるため、そうとは言い切れない。しかし、代数的関数の描画に関しては、数式処理アルゴリズムは十分な効力がある。

2．2　既存する数式処理システム

一般的によく知られている数式処理システム〔1〕として、

Macyma 1966年、M.Martin,J.Moses 他（MIT）

Reduce 1971年、A.Hearn（ユタ大学）

Maple 1981年、B.Char 他（カナダ・ウォータールー大学）

Mathematica 1988年、S.Wolfram 他

Risa/Asir 1990年、野呂正行 他（富士通株式会社国際研）

などがある。ただし、年号は開発年度、人名は開発者、大学等は開発時の所属である。

2．3　数式処理システムと関数描画の関係

関数描画システムよりみれば、数式処理システムは、数式の変形、簡素化を行う上で必要欠くべからざる要素である。確かに、数式処理システムの中には、多項式の関数描画ができるものがあるが、それらは数式処理システムの重要な部分を占めるのではなく、付加的なものとして扱われている。よって、関数描画はできるのであるが、それらの精度を考えたときには、必ずしも正確であるとは言いがたい。

関数描画は色々な問題を解くにあたっての重要な部分を占めており、この結果を用いて新たな問題を解く手段とすることが多いはずである。

現在では、数式処理システムはほぼ完成の域に達している。しかし、関数描画に関してはまだまだ完成しているとは言いがたい。そこで我々は、一歩先んじるためにより正確な関数描画を数式処理システムに組み込む必要があると考える。

我々が用いるシステムは、数々の計算機（ワークステーションやパソコンなど）で動作しており、C言語という比較的理解しやすいプログラム言語で作られており、だれにでも入手できる（インターネット上のいくつかのftpサイトから無償で入手可能）数式処理システムのRisa/Asir〔2〕である。

3　関数の描画とは

関数の描画とは、与えられた領域Dでの関数$f=0$の解を必要とされる出力装置（ディスプレイ、プリンター等）に出力することとする。

与えられた関数を表現する場合、関数の計算精度、描画精度は描画する装置の解像度に

依存する。そのため描画とは、関数と定義域および解像度と呼ばれるものによって定められる描画関数を計算することである。以下描画に対する基本的な概念として解像度ならびに描画関数χの定義を提案する。

3．1　定義

以下ここで扱う関数fは、R^nの連結部分集合D上で定義された関数であってD上連続な関数とする。

3．1．1　解像度

画像の解像度とは定義域Dの部分集合の族$S=\{S_i \mid S_i$は連結$i=1,\cdots, m\}$であって次のものをいう。

定義3．1．1　集合の族SがD上定義された関数fの解像度とは

$$D=\bigcup_{i=1}^{n} S_i, \quad S_i \cap S_j=\phi \quad i \neq j$$

ここで我々の関心は、ディスプレイやプリンターなどの出力装置に与えられた関数を描画することである。よって、定義域Dを出力領域と考え、集合Sを出力の最小単位であるドットと考えることができる。一般的に関数fはR^n上の任意の点で関数値を持つが、この解像度Sよりも小さな領域で関数fを考えることは出力装置の性質上から意味のないことである。

3．1．2　描画関数

ここで描画関数に関する定義を行う。基本的な考えは、出力装置に依存する解像度Sの中に$f(x)=0$となる解が存在するか否かを判定できるような関数が定義できればよいということである。

定義3．1．2　D上定義されている関数fの解像度Sによる描画関数（*character* $\chi(D, S, f)$）とは、

1. $\chi : S \rightarrow \{0,1\}$
2. $\chi(S_i)=0$ならばS_iの任意の元xに対し$f \neq 0$

定義３．１．３　D上定義されている関数fのSにおける*faithful character* χとは

1. χは関数fのSにおける*character*
2. $\chi(S_i)=1$ならばS_iのある元xが存在し$f=0$である

関数の描画としては、この*faithful character*を計算できれば問題は、解決している。しかし、関数が代数関数に限っても現在のところ万能な計算アルゴリズムは存在しない。そこで、十分実用に耐える*faithful character*を弱めた次のような描画関数*boundary character*を提唱したい。

定義３．１．４　D上定義されている関数fのSにおける*boundary character* χとは

1. $\chi : S \rightarrow \{0,1\}$
2. $\chi(S_i)=0$ならば$\overline{S_i}$の任意の元xに対し$f\neq 0$である。

ここで$\overline{S_i}$は集合S_iの境界からなる集合である。

我々の基本的な方針は全画面検索法を前提としている。そこで、全画面検索を行う最小単位として解像度Sを用い、描画関数χにより関数fの描画を行う。これにより、出力装置に依存する精度を満足しながら、関数描画を行うことができる。

4　現在の状況

ここでは、現在の関数描画に関する状況を報告する。

現在は、２次元空間（平面）上での関数$f(x, y)=0$の描画を全面検索法を用いて行っている。また、*faithful character*を用いるのではなく、十分実用に耐える*boundary character*を用いている。

利用している数式処理システムは、UNIX上で動作するRisa/Asirである。関数描画は、Risa/Asir に組み込まれているifplotというモジュールに改良を加えながら、ディスプレイ装置に対して行っている。

４．１　ifplot

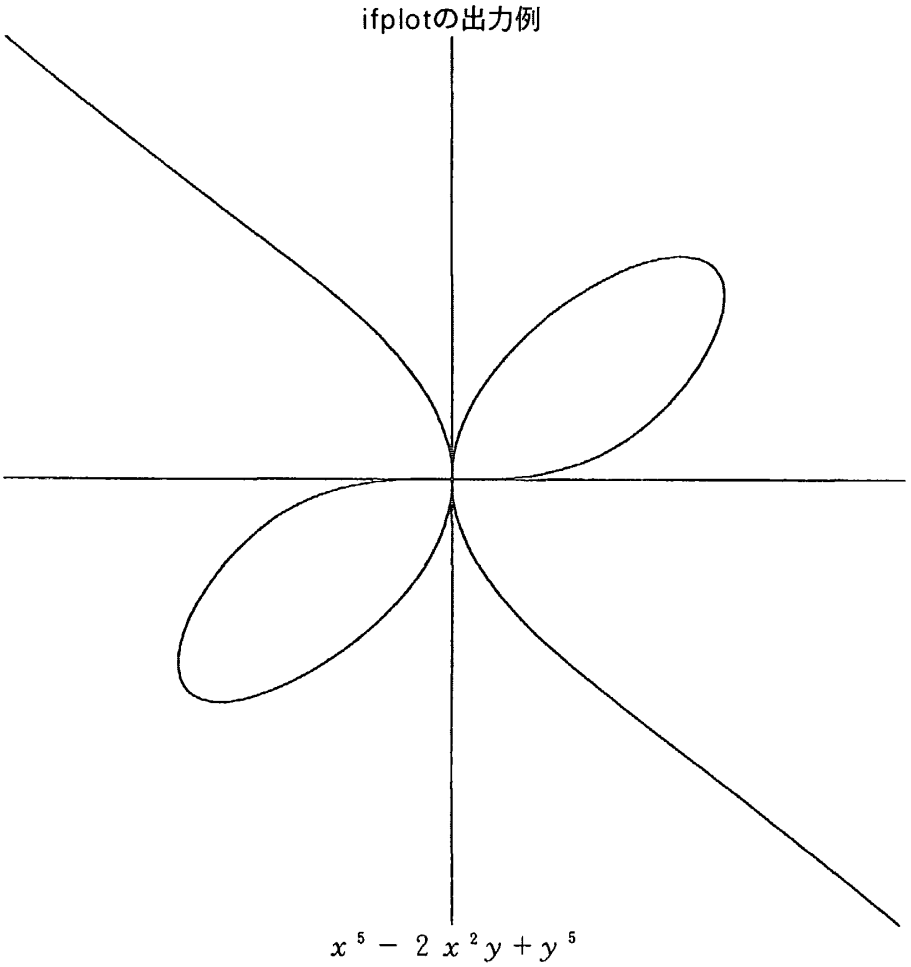
ifplotは、平面領域を小さな矩形に分割しその一つ一つを解像度S_iと捉えることにより、描画関数*boundary character*を求め関数描画を行っている。

ifplotの基本的な考え方は次の**中間値の定理**に基づいている。

定理4．1．1　中間値の定理〔3〕ある区間において連続な関数$f(x)$が、この区間に属する点a,bにおいて相異なる値$f(a)=\alpha$，$f(b)=\beta$をとるとき、α，βの中間にある任意の値をμとすれば、$f(x)$はa，bの中間のある点cにおいて、この値μをとる。すなわち、

$$a<c<b, f(c)=\mu$$

となるcが存在する。



解像度に対応する矩形の境界点$(x_0,y_0),(x_0,y_1),(x_1,y_0),(x_1,y_1)$の関数値$f(x,y)$の符号を判定することによって解像度$S_i$の*boundary character*を求めている。

例えば、$f(x_0,y_0)$と$f(x_1,y_0)$の符号が異なれば（$f(x_0,y_0)<0$と$f(x_1,y_0)>0$など）、中間値の定理によりx_0とx_1の区間内で必ず$f(x,y_0)=0$となる。

そして、*boundary character*により矩形領域の塗りつぶしを行えば、出力装置に依存する解像度で正確な関数描画を行うことができる。

4．2　ifplotの問題点

現在のifplotの関数描画は解像度に依存する範囲内において正確に行うことが可能であるが、厳密には正確であるとは言えない。

*boundary character*を求める時に中間値の定理を用いているが、その時の区間内に奇数個の解が関数に存在すれば良いのであるが（奇数個の解を持てば必ず区間の端の点の関数の符号は異なる)、偶数個の解が関数に存在する時が問題となる。すなわち、偶数個の解を持てば区間の端の点の関数値の符号は同じとなってしまい、中間値の定理を利用したこの方法では解を持たないと判断してしまうからである。

この問題の解決法として、Sturm の定理を利用する方法がある。

4．3　Sturm の定理の利用

ifplotの問題点を解決するための方法として、boundary characterを求める時に、Sturm 列を用いれば正確に区間内の解の個数を判定することが可能である。

定義４．３．１　多項式のSturm列〔４〕$f(x)$ を実係数方程式とする。ただし、$f(x) = 0$ は重根を持たないとする。従って、$f(x)$ とその導関数 $f'(x)$ は共通因数を持たない。

$f(x)$ と$f'(x)$ にユークリッドの互除法を行ってその最大公約数を求める。

$$
\begin{aligned}
f(x) &= q_1(x)\, f'(x) - f_2(x) \\
f'(x) &= q_2(x)\, f_2(x) - f_3(x) \\
f_2(x) &= q_3(x)\, f_3(x) - f_4(x) \\
&\cdots \\
f_{m-2}(x) &= q_{m-1}(x)\, f_{m-1}(x) - f_m
\end{aligned}
$$

ここで、f_mは 0 でない定数を表し、このようにして得られた整式の列

$f_0(x), f_1(x), f_2(x), \cdots, f_m$ （ただし、$f_0(x)=f(x), f_1(x)=f'(x)$）

のことを多項式のSturm列と定義する。

定理4．3．1　Sturmの定理［4］$a<b$でa, bは$f(x)$の根でないならば、aとbの間にある$f(x)$の実根の個数は$V(a)-V(b)$ に等しい。ただし、$V(a)$はSturm 列

$$f_0(a),\ f_1(a),\ f_2(a)\ \cdots,\ f_m$$

の符号の変化の数とする。

中間値の定理の場合は、ある区間の端の点の関数値の符号が異なればその区間内に必ず存在することが言えるが、この定理の場合は、Sturm 列を求めることにより“その区間内の解の個数までも判定することが出来るという”ことである。

しかしながら、この方法では“計算コストが非常に大きくなる”という欠点がある。すなわち、中間値の定理の場合は、区間の端の点の関数値を求めるだけで良かったが、Sturm 列の場合は、ユークリッドの互除法によりSturm 列を求め、その各々の関数値を求めなければならない。よって、今後の課題としては、この問題の解決を図るためにSturm 列を利用する以外の新たなアルゴリズムを構築する必要がある。

注　現バージョンのifplotには、Sturmの定理を用いて関数描画を行うオプションスイッチがついているが、この場合の関数描画に要する時間は、Sturmの定理を用いない場合に比べて数倍以上のひらきがある。

5　今後の課題

関数描画に関しては、*boundary character*の場合の理論的には全て解決していると言っても良いが、実際問題としては計算機上で関数描画を行おうとすると、実用に耐えうる範囲内では必ずしも解決しているとは言えない。

すなわち、理論的背景を厳密にして関数描画を行っても、命令を入力してから結果が得られるまでの時間が果てしないものであるとすると、それは使いものにならないものである。

今後の課題として計算量のより少ないアルゴリズムの開発が必要と考える。

参考文献

〔1〕渡辺隼郎．数式処理とは．数式処理入門から高度利用まで．archive No.12.p.16 - p23．ＣＱ出版社，1990/8/20

〔2〕野呂正行，下山武司．Asir User's Manual Edition 2.0 for Asir - 940420. ftp from ftp.mm.sophia.ac.jp,1994

〔3〕高木貞二．解析概論改訂第三版．岩波書店，1971

〔4〕淡中忠郎．代数学．朝倉数学講座1.朝倉書店，1974